DOCUMENT RESUME

ED 112 865 IR 002 574

AUTHOR	Mayer, Richard E.
TITLE	Instructional Variables in Computer Programming. Indiana Mathematical Psychology Program. Final Report.
INSTITUTION	Indiana Univ., Bloomington. Dept. of Psychology.
SPONS AGENCY	National Science Foundation, Washington, D.C. Office of Experimental Projects and Programs.
PUB DATE	31 Aug 75
NOTE	65p.
EDRS PRICE	MF-$0.76 HC-$3.32 Plus Postage
DESCRIPTORS	*College Students; Computer Assisted Instruction; *Computer Science Education; Higher Education; Instructional Media; *Learning Characteristics; *Programing; Research Projects; Teaching Methods
IDENTIFIERS	Aptitude Treatment Interaction; Instructional Variables; Model Learning; Practice; Rule Learning

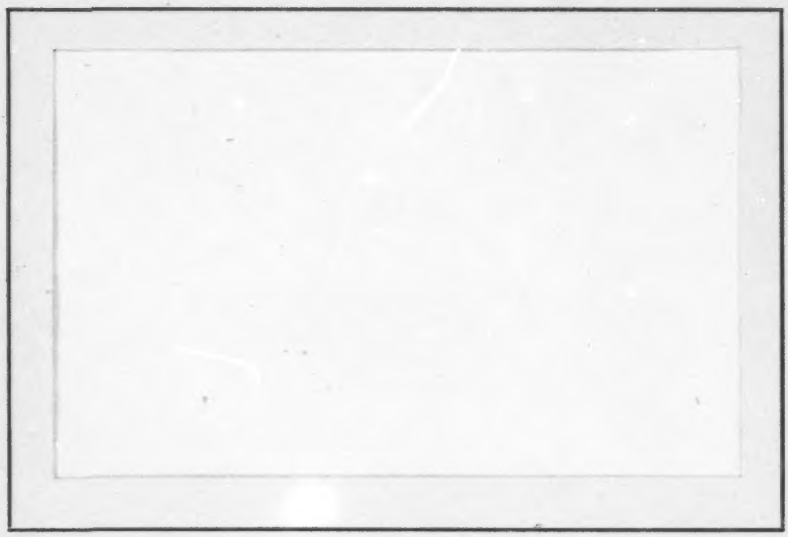
ABSTRACT

The final report of this study describes the objectives and plan of attack used for determining how novice students learn to interact with a computer and how instruction can result in meaningful learning. Changes to the original plans and significant outcomes are mentioned. The final report concludes with abstracts of research findings on the effects of models, the effects of practice questions and aptitude effects of program representations, and effects of computational vs. meaningful practice in programed instruction. A list of papers published and delivered under this study is provided. "Instructional Variables in Meaningful Learning of Computer Programing," a paper from the study, is appended to the final report. Additional appendixes contains the text of the model and rule booklets, typical practice and test questions and a pretest used in the study. (CH)

****
* Documents acquired by ERIC include many informal unpublished *
* materials not available from other sources. ERIC makes every effort *
* to obtain the best copy available. Nevertheless, items of marginal *
* reproducibility are often encountered and this affects the quality *
* of the microfiche and hardcopy reproductions ERIC makes available *
* via the ERIC Document Reproduction Service (EDRS). EDRS is not *
* responsible for the quality of the original document. Reproductions *
* supplied by EDRS are the best that can be made from the original. *
****

ED112865

Indiana Mathematical Psychology Program



Department of Psychology
Indiana University
Bloomington, Indiana 47401

IR 002 574

IR

FINAL REPORT

Project No. EPP74-11479

Grant No. EC-44020

June 1, 1974 to August 31, 1975

Instructional Variables in Computer Programming

Richard E. Mayer

Indiana University

Bloomington, Indiana 47401

August 31, 1975

National Science Foundation

Office of Experimental Projects and Programs

Technological Innovation in Education

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

A. Objectives and Plan of Attack

Although it seems clear that computer programming will play an increasing role in education, little is presently known concerning how novice students learn to interact with a computer (Weinberg, 1971; Papert, 1971; Miller, 1971) nor how to develop technical instruction which results in meaningful learning (Mayer, 1972). During the past year, some attention has been directed towards proposing or conducting preliminary studies of the cognitive processes involved in learning a computer programming language by novices (Sime, Green & Guest, 1973; Weisman, 1974; Gould, 1974; Kreitzberg & Swanson, 1974; Shniederman & Ho, 1974).

The present study attempted to provide a further modest step by conducting a series of laboratory studies concerning how meaningful learning is influenced by a diagram model of the computer which is expressed in familiar terms, by various types of practice exercises, and by the learner's abilities.

One factor in the acquisition of new technical information is the availability of a body of existing, familiar experience in memory -- a meaningful learning set -- which may be used during learning to assimilate new material (Ausubel, 1968; Piaget, 1970; Brownell, 1935). In particular, the work of David Ausubel (1968) points to the importance of a learner's assimilative set as a determinant of the learning outcome; his distinction between "meaningful" vs. "rote learning sets" as an internal factor influencing learning has added a new -- albeit not fully understood -- dimension to the problem of providing instruction which results in meaningful learning.

The present study provides some information on the question of what kind of pre-requisite knowledge must be available in a learner's memory before and during instruction. More specifically, do subjects who have the model available during learning perform differently on a posttest than those who do not?

A second important factor is that the pre-requisite experience not only be available during learning, but that it is actively processed and used during learning (e.g., Roughead & Scandura, 1968; Gagne & Brown, 1961). The work of Rothkopf and his associates (Rothkopf, 1970) has pointed, for example, to the importance of question answering activity during learning as a "mathemagenic activity"; one aspect of this, as of yet poorly understood, concept seems to be the activation of existing experience which can be related to new material.

The present study provides information on the question of how to activate pre-requisite knowledge so that new material may be assimilated to it during learning. Specifically, does the amount or type of practice questions asked during learning influence posttest performance, and does it have different effects under different instructional treatments?

The need to investigate the availability and the activation of existing assimilative sets as possible conditions for meaningful learning was summarized by Mayer & Greeno (1972, p. 166);

> ... different instructional procedures could activate different aspects of existing cognitive structure. And since the outcome of learning is determined by the new material and the structure to which it is assimilated, the use of different procedures could lead to the development of markedly different structures during learning of the same concept.

A final question, suggested by investigations of aptitude x treatment interaction (Cronback & Snow, 1969; Bracht, 1970) concerns whether learners with different ability levels learn better under different instructional methods. For example, do high mathematics ability learners already possess a "meaningful learning set" which may conflict with the model while low ability learners possess very little relevant knowledge and are aided by the model?

B. Changes to the Original Plans

Each of the studies in the original proposal, and outlined above, has been completed and full reports of the work are available in the form of a separate technical report, "Instructional Variables in Meaningful Learning of Computer Programming", Indiana University Mathematical Psychology Program, Report No. 75-1, 1975. The studies focused on the effects of diagram models of the computer as an aid in learning, the role of different types of practice in learning programming, and the effects of learners' aptitudes on the amount and quality of learning.

In addition, as cited in the original proposal, several supplemental studies were completed which compliment the main studies. These supplemental studies involved effects of program representation, e.g., how to develop a computer programming language that is easy for novices to understand, and effects of computational vs. meaningful practice in programmed instruction, e.g., what type and how much practice should be used in teaching technical information to novices.

C. Significant Outcomes

The experiments conducted under this grant provided several important new contributions to our understanding of how non-programmers learn computer programming and the results have implications for the design of instruction of technical information for non-professionals. These findings are particularly important in light of the increasing need of non-programmers and non-professionals to interact with computers and other technical systems, including the increasing use of computer technology in education.

The main results were: (a) A model which presents the computer as a familiar analogy (e.g., eraseable scoreboard for the computer's memory) presented prior to learning results in better ability to write long programs and understand written programs while no experience with the model results in a much more specific learning of how to write programs like those in the instructional text. (b) Practice on writing simple programs aids subjects given the model, while practice on interpreting programs (requiring more thinking) aids non-model learners most. (c) Performance on pretests is correlated to specific types of errors in learning and thus may be used as an aid in determining where to emphasize instruction for different students. (d) Subtle differences in the conventions of the programming language have large effects for novices. (e) Rigid, computational practice may result in learning that limits the learner's ability to transfer to new situations.

Abstracts of these findings are given below.

Effects of models, practice questions and aptitude.

One hundred seventy six non-programmers learned a computer programming language either within the context of a diagram model of a computer expressed in familiar terms or with no model, and then practiced on exercises and took a posttest. In learning and posttest performance, Model Ss excelled on straightforward generation of programs. The model was especially helpful for low ability Ss. Practice in interpretation helped Non-Model Ss most and practice in writing simple programs helped Model Ss most. The roles of the model in establishing a meaningful learning set, and of practice on mathemagenic activity, were discussed.

Effects of Program Representation.

A program-like branching system describing what prizes (A through F) were awarded for particular outcomes of a tournament of games among three teams were presented to 200 subjects as either a verbal list with "go to" structure (Jump), a shortened verbal list (Short-Jump), nested verbal paragraphs with "if ... then ... else" structure (Nest), a matrix table (Example), or as a diagrammatic representation of each of these. In tests of comprehension, the overall performance increased from lowest to highest as follows: Jump, Short-Jump, Nest, Example; and this order was particularly strong for performance on complex questions relative to less complex questions. Jump and Short-Jump performance was relatively higher with diagrams and Example was lower with diagrams. Implications for a theory of problem representation and for development of computer programming languages were discussed.

Effects of computational vs. meaningful practice in programmed instruction.

Eighty subjects read six passages that were supplemented by questions asking for the following: formal definitions (Definition), calculating a value (Calculating), applying a conceptual model to a problem (Model), all three (All), or no questions (None). Results on tests containing all three question types given after Passages 7 and 8 indicated an overall superiority of Group All over Group None, a Treatment X Question Type interaction in which Group Model excelled on all questions but Groups Calculating and Definition did not, and no difference between subjects who had answered questions and those who simply read them in earlier passages. Implications for the acquisition processes were discussed.

D. Published and Delivered Papers

The following papers report the work conducted under this grant and acknowledge N.S.F. support.

Effects of models, practice questions and aptitude.

Mayer, R.E. Different problem solving competencies established in learning computer programming with and without a meaningful model. Journal of Educational Psychology, accepted and in press.

Mayer, R.E. Instructional variables in meaningful learning of computer programming. Indiana University: Indiana Mathematical Psychology Report Series, Report 75-1, 1975. Included as Appendix

Mayer, R.E. Instructional variables in meaningful learning of computer programming. Paper read as annual meeting of American Educational Research Association, April, 1975.

Effects of program representation.

Mayer, R.E. Comprehension as affected by structure of problem representation. Memory and Cognition, accepted and in press.

Effects of computational vs. meaningful practice.

Mayer, R.E. Forward transfer of different reading strategies evoked by testlike events in mathematics text. Journal of Educational Psychology, 1975, 67, 175-169.

Mayer, R.E. Do practice problems and objectives limit a subject's learning set? Paper read as annual meeting of American Psychological Association, September, 1975.

Appendix

Instructional Variables in Meaningful Learning of Computer Programming[1]

Richard E. Mayer[2]

NSF-TIE-EC44020

Report No. 75-1

PREFACE

This is a report of a program of research on instructional variables in computer programming, supported by the National Science Foundation, under NSF Grant EC-44020 to the author, and monitored by the Office of Experimental Projects and Programs, Technological Innovations in Education.

This work was conducted at the Department of Psychology of Indiana University. Thanks are due to Cristine Ward-Hull and Don Young who assisted in the collection of data. The helpful comments and encouragement of Frank Restle and the assistance of the staff of the Cognitive Institute of Indiana University are greatly appreciated.

Instructional Variables in Meaningful Learning of Computer Programming[1]

Abstract

One hundred seventy six non-programmers learned a computer programming language either within the context of a diagram model of a computer expressed in familiar terms or with no model, and then practiced on exercises and took a posttest. In learning and posttest performance, Model Ss performed best on problems requiring interpretation while Non-Model Ss excelled on straightforward generation of programs. The model was especially helpful for low ability Ss. Practice in interpretation helped Non-Model Ss most and practice in writing simple programs helped Model Ss most. The roles of the model in establishing a meaningful learning set, and of practice on mathemagenic activity, were discussed.

Although it seems clear that computer programming will play an increasing role in education, little is presently known concerning how novice students learn to interact with a computer (Weinberg, 1971; Papert, 1971; Miller, 1971) nor how to develop technical instruction which results in meaningful learning (Mayer, 1972). During the past year, some attention has been directed towards proposing or conducting preliminary studies of the cognitive processes involved in learning a computer programming language by novices (Sime, Green & Guest, 1973; Weisman, 1974; Gould, 1974; Kreitzberg & Swanson, 1974; Shniederman & Ho, 1974).

The present study attempted to provide a further modest step by conducting a series of laboratory studies concerning how meaningful learning is influenced by a diagram model of the computer which is expressed in familiar terms, by various types of practice exercises, and by the learner's abilities.

One factor in the acquisition of new technical information is the availability of a body of existing, familiar experience in memory -- a meaningful learning set -- which may be used during learning to assimilate new material (Ausubel, 1968; Piaget, 1970; Brownell, 1935). In particular, the work of David Ausubel (1968) points to the importance of a learner's assimilative set as a determinant of the learning outcome; his distinction between "meaningful" vs. "rote learning sets" as an internal factor influencing learning has added a new -- albeit not fully understood -- dimension to the problem of providing instruction which results in meaningful learning.

The present study provides some information on the question of what kind of pre-requisite knowledge must be available in a learner's memory before

and during instruction. More specifically, do subjects who have the model available during learning perform differently on a posttest than those who do not?

A second important factor is that the pre-requisite experience not only be available during learning, but that it is actively processed and used during learning (e.g., Roughead & Scandura, 1968; Gagne & Brown, 1961). The work of Rothkopf and his associates (Rothkopf, 1970) has pointed, for example, to the importance of question answering activity during learning as a "mathemagenic activity"; one aspect of this, as of yet poorly understood, concept seems to be the activation of existing experience which can be related to new material.

The present study provides information on the question of how to activate pre-requisite knowledge so that new material may be assimilated to it during learning. Specifically, does the amount or type of practice questions asked during learning influence posttest performance, and does it have different effects under different instructional treatments?

The need to investigate the availability and the activation of existing assimilative sets as possible conditions for meaningful learning was summarized by Mayer & Greeno (1972, p. 166);

> ... different instructional procedures could activate different aspects of existing cognitive structure. And since the outcome of learning is determined by the new material and the structure to which it is assimilated, the use of different procedures could lead to the development of markedly different structures during learning of the same concept.

A final question, suggested by investigations of aptitude x treatment interaction (Cronback & Snow, 1969; Bracht, 1970) concerns whether learners

with different ability levels learn better under different instructional methods. For example, do high mathematics ability learners already possess a "meaningful learning set" which may conflict with the model while low ability learners possess very little relevant knowledge and are aided by the model?

EXPERIMENT I

In Experiment I, novice programmers learned a simple computer programming language from text that either presented a pictorial model of the computer and explained how the computer model functioned in relation to each programming statement (Model Group) or one which provided identical definitions and examples of each statement but without any model (Rule Group). Half the subjects in each group received, as part of their texts, an example program which exemplified each of the to-be-learned statements (Program Aid). Following instruction, all subjects practiced with feedback and to a criterion of four in a row correct on two successive exercise sets. Both sets contained exercises in writing a program to solve a given problem (Generation Problem) and in interpreting what a given program would do (Interpretation Problems), but Set 1 dealt only with one line statements, and Set 2 with non-looping programs. Following practice, subjects received a transfer posttest consisting of six generation and six interpretation problems, given with feedback but not to any criterion, and which required a "looping" program.

Method

Subjects and design The subjects were forty Indiana University students who participated in the experiment in order to fulfill a requirement for their introductory psychology course. The main between subjects factor was

instructional method (Model vs. Rule), but half the subjects in each group received an additional example of how the statements went together to form a program (Program vs. No Program) and half the subjects in each sub-group scored high in Math SAT and half scored low (High vs. Low Ability). Every subject received the same two practice sets and the same posttest set with two types of questions in each (Generation, Interpretation), so the comparison across type of problem was a within subject comparison.

Materials. In order to teach a simplified version of FORTRAN, two main instructional booklets and three problem sets were constructed. Both booklets presented the traditional definitions and examples of seven programming statements (READ, WRITE, GO TO, IF, STOP, Arithmetic Statements, Counter Set Statements) and the appropriate grammatical rules (e.g., counter names, pointer names, format notation). Booklets were organized in the following manner.

The Model text began with a diagram of the inside working components of the computer showing "input window" (described as a ticket window), "output pad" (described as a pad of message paper), "memory scoreboard" (described as an eight space, erasable scoreboard), and "program list with pointer arrow" (described as a shopping list). Each of seven programming statements was defined, exemplified and explained within the context of the diagram with an attempt to help the learner "role play" what the computer does for each statement. For example, the statement

P6 GO TO P4

could be related to the model by noting that the pointer arrow would move from the sixth statement to the fourth statement on the list and the computer would do whatever P4 says. Similarly, the statement

A3 = 0

could be related to the model by noting that the computer would erase whatever it had on the memory scoreboard at space A3 and write in zero instead. The organization of the Model text was as follows: diagram and explanation of the diagram, input-output functions (giving definitions and examples of READ and WRITE), memory functions (giving definitions and examples of Arithmetic and Counter Set statements), and program control functions (giving definitions and examples of GO TO, IF and STOP).

Rule text contained the seven statements defined and exemplified on seven separate pages but there was no conceptual framework added. The organization was: brief introduction, definition and examples of READ, WRITE, Counter Set, Arithmetic, GO TO, IF and STOP statements. The definitions and examples were identical in all booklets, and only the conceptual framework (Model)was varied.

Half the subjects in each group received an additional page at the front of their booklet (Program Aid) which presented an example seven line program -- containing each of the seven types of programming statements. Instructional texts with the Program Aid presented the statements in the order in which they appeared in the example program (Counter Set, READ, IF, Arithmetic, GO TO, WRITE, STOP).

A series of three sets of exercises were constructed. Practice Set 1 consisted of 24 single statements, Practice Set 2 consisted of 24 non-looping programs, and the Posttest set consisted of 12 looping programs. For each exercise item, a statement of the problem in English was typed on one 3x5 card, and the correct program to solve the problem was typed in computer language on the other side.

Thus in each exercise set two types of questions could be asked for each item. Generation Exercises gave a problem in English and asked the subject to write a program to solve it. An example from Set 1 (Statements) is: "Given a number is in memory space A6, write a statement to increase that number by 1." An example from Set 2 (Non-Looping Program) is: "Given that a card with a number on it is input, write a program to print out that number unless it is zero." An example from the Posttest Set (Looping Program) is: "Given a pile of data cards with a number on each is input, write a program to print out the square of each number and to stop when it gets to card with a zero on it."

Interpretation Exercises, on the other hand, presented a program and asked the subject to write in English what problem it would solve. For example, in Set 1, given the statement,

```
A6 = A6 + 1
```

S would write the problem that it solves (see first example above). An example from Set 2 is, given the program,

```
P1 READ(A1)
P2 IF(A1=0) GO TO P4
P3 WRITE(A1)
P4 STOP
```

write the problem it solves (see second example above). An example from the posttest is to write the problem (see third example above) which is solved by the program,

```
P1 READ(A1)
P2 IF(A1=0) GO TO P6
P3 A1=A1*A1
P4 WRITE(A1)
P5 GO TO P1
P6 STOP
```

In addition, materials included a Pre-Experimental Questionnaire asking about the subject's experience with computer programming and asking for his MSAT score, a Pre-Test consisting of six algebra problems (e.g., $Y^2 - 6 = 10$. Find Y.) and answer sheets for the exercises.

Procedure. Subjects were run in small groups of 2 to 4 with several different instructional treatments represented within each session. First all subjects took an algebra pretest and filled out a short questionnaire which asked for math SAT score and which solicited information concerning experience with computer programming. When all subjects were finished, instructions were read with subjects again being asked to indicate familiarity with computer programming, and the instructional booklets were passed out. Subjects were told to read through the booklets at their own rates and to "try to understand what it is talking about" so as to be prepared for a "test".

When a subject finished reading his booklet, the procedure for the three exercise sets was explained and the subject was given an answer sheet and a pile of exercise cards for Set 1. The items were randomly arranged except for the constraint that the type of problem (i.e., Generation or Interpretation) alternated for each item. No two items were identical. The subject was instructed to pick the first card; if it stated a problem, to write a program to solve it, and if it stated a program, to write a problem it would solve; to then flip over the card to find the correct response, to write a "C" if he was correct and to write the correct answer on the right side of the answer sheet if his answer was not correct, and to then go on to the next card and so on. When the subject correctly answered four problems in a row (i.e., two exercises of each type) he went on to Set 2, and when he correctly

answered four items in a row in Set 2, he went on to the posttest which he continued for all 12 items with feedback on each.

Results

Three subjects indicated previous experience with computer programming, four subjects were unable to solve 3 of the 6 problems on the algebra pretest, and 2 subjects were unable to reach criterion on Set 1 or Set 2 within 2 hours. Data for these subjects were eliminated and new ones were run in their places. Subjects indicating MSAT scores of 560 or above were counted as High Ability and those with scores below 560 were counted as Low Ability.

Each subject's response for each exercise item was scored as either correct or incorrect. Interpretation answers conveying the correct initial condition (one card, two cards, or a pile of cards) and stating the problem in a sentence (e.g., "count the number of cards until a 99 appears") that would lead to writing the target program were scored as correct. In the posttest, step-by-step translations of each statement, or responses which failed to express the fact that a pile of cards and looping was involved were counted as incorrect. Generation answers were counted correct if they contained the right statements in the right order even if format errors (e.g., WRITE A1 instead of WRITE (A1)) or grammar errors (e.g., GO TO P3 IF (A1=3) instead of IF (A1=3) GO TO P3) were present. In the posttest, the most frequent errors that resulted in being counted as incorrect were a failure to include a GO TO statement that would allow looping for Generation items, and a failure to indicate a pile of cards rather than just one card was involved in Interpretation items.

Since the posttest consisted of items similar to those commonly used in programming workbooks, and since all subjects had reached a reasonable criterion of learning on the seven basic programming statements before beginning the posttest, the main interest of this experiment was whether subjects' initial instructional booklet would influence transfer of learning of the posttest. The proportion correct response on the posttest by type of exercise for each instructional group is given in Table 1, and an analysis of variance was performed on the data. There was no main effect due to presence of Model Text ($F(1,32) < 1.00$, p=n.s.) but there was a reliable negative effect due to presence of the Program Aid ($F(1,32) = 5.80$, $p < .025$). Apparently, introducing a complex program before subjects were familiar with the basic statements confused subjects and made it more difficult for them to acquire the new statements in generalizable form. The only other reliable effect is a Model x Type of Problem interaction ($F(1,32) = 5.67$, $p < .025$) with subjects who received instruction that involved the diagram model (Model Text) performing better than subjects who did not (Rule Text) on Interpretation items but worse on Generation items. These results support the idea that subjects in the two instructional groups acquired learning outcomes that differed in structural or qualitative ways that cannot be explained in terms of transfer of specific information in the text.

There was also a slight tendency for instruction that involved the model to improve performance for low ability subjects compared with non-model texts (48% correct vs. 42% correct overall, respectively) and the reverse was true for high ability subjects (51% vs. 53% correct overall, respectively); however, the Model x Ability interaction failed to reach even a marginally reliable level ($F(1,32) < 1.00$, p = n.s.).

Table 1

Proportion Correct Response on Transfer Posttest for Two Instructional Groups by Type of Problem

Instructional Group	Problem Type	
	Generation	Interpretation
Rule	.47	.11
Model	.27	.27

Note. - Main effect of instructional treatment, p = n.s.; treatment x type of problem interaction, $p < .025$.

When performance, i.e., number of errors, was summarized over all three sets of exercises, the Model x Type of Exercise interaction was retained ($F(1,32) = 6.97$, $p < .025$) with Model subjects averaging less errors than non-Model subjects on Interpretation items (8.6 vs. 10.5 respectively) and averaging more errors on Generation items (8.4 vs. 6.5 respectively). In addition the Model x Ability interaction reached statistical reliability ($F = 4.23$, df - 1/32, $p < .05$) with Model Texts resulting in fewer errors for low ability Ss than non-Model Texts (7.8 vs. 10.3 respectively) but the reverse is true for high ability Ss (9.2 vs. 6.9 errors respectively). Thus there is mild support for the suggestion that high ability subjects already had their own "models" while low ability subjects did not.

Table 1 about here

EXPERIMENT 2

In Experiment 2, each subject read an instructional booklet and then took an 18-item posttest. In addition to the Model Text and the Rule Text used in the previous study, Experiment 2 also involved two new instructional booklets -- Flow Text which introduced flow chart symbols and explained each statement in the context of writing a flow chart, and Both Text which contained both the Model and the Flow booklets. Half the subjects in each instructional group were given practice with feedback in writing and interpreting statements, and non-looping programs and half the subjects in each group received no practice. All subjects were given the same posttest, consisting of generating and interpreting statements, non-looping programs and looping programs, and unlike the previous studies subjects received no feedback on the posttest.

Method

Subjects and design. The subjects were 80 Indiana University students drawn from the same population as in Experiment 1. Each subject served in one cell of a 4 x 2 x 2 factorial design, with the first factor being method of instruction (Model, Rule, Flow, Both), the second factor being general mathematical ability (High MSAT score vs. Low MSAT score) and the third factor being amount of practice in learning (Practice, vs. No Practice). Since all subjects received the same posttest, comparisons across type of posttest item were within subject comparisons.

Materials. Two of the four instructional booklets were the Model and the Rule Texts (without "program aid") used in the previous study. In addition, the Flow Text was nearly identical to the Rule Text except that an introductory page presented each of the basic flow chart symbols with an example flow chart containing all of the seven to-be-learned statements, and each statement was discussed in the text within the context of where it fit in the flow chart. Finally, the Both Text contained both the Flow and the Model booklet.

A deck of eight practice cards (with correct answers on the back of each) was constructed consisting of two generation and two interpretation items selected from Set 1 used in the previous study and two generation and two interpretation items selected from Set 2. An 18-item posttest deck was also constructed using a 2 x 3 design, with the first factor being type of problem (Interpretation or Generation) and the second factor being complexity of problem (Statement, Non-Looping Program, Looping Program). There were three problems for each cell, each typed on a 3 x 5 card as in previous

experiment but with no indication of the correct answer, and with all items different from those given in the practice deck.

The same Pre-Experimental Questionnaire and Pre-Test were used as in Experiment 1.

Procedure Subjects were run in small groups of 2 to 4, as in Experiment 2. First subjects completed the pre-experimental questionnaire. Then instructions were passed out and subjects were given instructional booklets as in Experiment 1. When the subject finished reading, he was given the deck of eight practice problems and an answer sheet to work on if he was in the Practice Group. Subjects received feedback but there was no learning criterion. When the subject finished all eight practice items, or if he was in the No Practice Group, he was given an answer sheet and 18 problem cards for the posttest. Unlike the previous experiment, no feedback (i.e., no knowledge of correct response) was given on the posttest. After reading the booklet and again at the end of the experiment, each subject was asked to indicate familiarity with the concepts presented.

Results

The posttest performance was scored as in the previous experiment. As in Experiment 1, data for subjects who indicated familiarity with computer programming were eliminated from the experiment (N=5); however, subjects scoring low on the pretest were retained. For purposes of an ANOVA, subjects in each of the four treatment groups were divided into low general math ability (MSAT below 560) and high general math ability (MSAT of 560 or above).

Table 2 shows the performance of the two instructional groups on each of the six kinds of posttest items. There was no overall effect due to practice ($F < 1.00$) and all interactions involving this factor produced F values less than 1; therefore, the data for practiced and non-practiced subjects has been merged. Practice may have had no effect in Experiment 2 because it was on very simple (and few) problems and not maintained to criterion, whereas practice was far more sophisticated in the previous experiment.

There was an overall superiority of subjects who received the model either in the Model Text or in the Both Text over those who did not ($F(1,72) = 9.15$, $p < .01$); however, as in previous experiments there was no overall difference between the Model and the Rule groups ($F(1,32) = 1.32$, p = n.s.) thus frustrating the question of which method is best. As in Experiment 1 there was reliable interaction involving method of instruction and type of posttest problem, with the two groups that received the model excelling on Interpretation items and the two groups not receiving the model excelling on Generation items ($F(1,72) = 7.63$, $p < .01$). In addition there was reliable three way interaction involving method of instruction, type of problem, and complexity of problem, in which the groups exposed to the model excelled especially on far transfer such as Looping Generation and Non-looping Interpretation and the groups not exposed to the model excelled on the most straightforward problems, the Non-looping Generation items ($F(2,144) = 6.96$, $p < .01$). These interactions suggest, as in Experiment 1, that instruction which involved pre-exposure to a model and reference to it during learning resulted in a qualitatively different learning outcome than non-model methods.

Table 2

Proportion Correct Response on Transfer Posttest

by Type and Complexity of Problem

Instructional Group	Type and Complexity of Problem					
	Generation			Interpretation		
	State-ment	Non-Looping Program	Looping Program	State-ment	Non-Looping Program	Looping Program
Rule	.67	.52	.12	.42	.32	.12
Model	.63	.37	.30	.62	.62	.09
Flow	.55	.48	.18	.18	.05	.05
Both	.78	.67	.25	.60	.42	.13

Note. - Main effect of Model, p = n.s.; Model x Problem Type interaction, $p < .05$; Model x Problem Type x Problem Complexity interaction, $p < .001$. Main effect of Flow, p = n.s.; Flow x Problem Type interaction, $p < .01$; Flow x Problem Type x Problem Complexity, $p < .025$.

Comparisons between the two groups which received the Flow Chart aid (Flow and Both Text) and the two groups which did not (Rule and Model Text) indicated a two way interaction involving method of instruction and type of problem ($F(1,72) = 11.65$, $p < .01$) and three way interaction involving method of instruction, type of problem and complexity of problem ($F(2,144) = 4.69$, $p < .025$). However, these interactions suggest that the flow chart aid -- unlike the model aid -- resulted in poorer performance on far transfer items such as Interpretation items or Looping items and better relative performance on near transfer such as Generation and Non-looping items. Apparently, the Flow Chart Aid, like the Program Aid in Experiment 1 restrained the depth of encoding of new information.

Table 2 about here

Although there was an overall superiority of subjects scoring high in MSAT over subjects scoring low ($F(1,72) = 6.00$, $p < .025$), the interaction between general mathematics ability and instructional method which was found in the previous experiment was not present in Experiment 2 ($F < 1.00$). This may be due to the fact that in Experiment 2, subjects were given no feedback on the correctness of their responses on the posttest, while in Experiment 1 the feedback which was given may have helped most the Low Ability Rule Text subjects, i.e., subjects who by virtue of their instruction and previous experience had the most to gain from this manipulation.

Supplemental Analysis of Experiment 2

In order to determine the role of a learner's ability, four tests were given to subjects in Experiment 2 prior to instruction. The four tests were

as follows: (a) Algebra Computation Test consisted of six items which asked for solutions to formal algebraic equations, e.g., "If X = Y + 2C, Y = A - B, Find the value of X in terms of A, B and/or C." (b) Algebra Story Test consisted of eight story problems which asked for the production of a solution formula, e.g., "A car rental service charges eight dollars a day and five cents a mile to rent a car. Find the expression for total cost C, in dollars, of renting a car D days to travel M miles." (c) Organization of Permutations Test consisted of asking the subject to write all possible arrangements of 1234 as described by Leskow and Smock (1970). (d) Organization of Cards Test presented four Katona (1940) card trick problems, e.g., "Suppose I had a deck of six cards, half red and half black and that I alternately place one card on the table and one on the bottom of the deck. If the cards appeared on the table in the order R,B,R,B,R,B what was the original order of the six card deck?"

The Organization of Permutations Test was scored by a manner sensitive to how orderly or systematically the subject produced permutations (see Leskow and Smock, 1970). The other tests were scored by a straightforward (strict) key with answers counted either as correct or incorrect. Separate analyses of variance failed to reveal a reliable pattern of aptitude x treatment interaction for any of the tests -- including the MSAT score -- thus suggesting that the ability level effected performance under each instructional treatment in similar ways.

In order to determine the relationships between each pretest measure and performance on each kind of posttest item, correlation coefficients were computed for each instructional group (N = 20) and for all subjects (N = 80).

Since no reliable differences were obtained among the groups on key correlation coefficients, only the grouped data is shown in Table 3. For this sample size, correlations above $r = .22$ are reliable at $p < .05$ and correlations above $r = .30$ are reliable at $p < .01$. As might be expected all five pretest measures correlated positively and reliably with overall posttest score. Since the MSAT is a general test of many types of mathematical reasoning ability it is not surprising that it correlates at moderate levels. with performance on all six kinds of posttest tasks, although it is interesting to note that it correlates more highly with Generation than Interpretation items. Apparently, the strongest overall pretest predictor in this study was the Algebra Story Test -- a test requiring subjects to translate story problems into formulas. This skill seems closely related to skills required in computer programming in general and the high correlations with performance for all six posttest tasks supports this view.

The other three tests appear to be more specialized and therefore of value in diagnosing potential learning difficulties. The Algebra Solution Test -- requiring straightforward solution of formal algebraic equations -- correlates well only with the most straightforward posttest tasks, i.e., memory of statements and generation of simple non-looping programs. The two tests which require more interpretation and some concept of looping in solving a problem -- the Permutations and the Card Tests -- correlate poorly with the posttest tasks that the Algebra Solutions Test correlates with, and correlate more strongly with far transfer items such as Looping items.

Table 3 about here

Table 3

Correlations Among Pretest and Posttest Scores

for All Subjects (N = 80)

Pretest Variables	Posttest Variables						
	Generation			Interpretation			TOTAL Posttest Scores
	State-ment	Non-Looping Program	Looping Program	State-ment	Non-Looping Program	Looping Program	
MSAT	.39	.37	.31	.26	.17	.30	.45
Algebra Solution	.40	.26	.07	.36	.11	.19	.34
Algebra Story	.46	.52	.33	.46	.30	.35	.61
Permutations	.05	.11	.29	.16	.20	.13	.23
Cards	.14	.21	.25	.32	.27	.42	.39

Note - All correlations are positive. For N = 80, $r > .22$ is reliable at $p < .05$; $r > .30$ is reliable at $p < .01$.

EXPERIMENT 3

Experiment 3 used the Model Text and the Rule Text as in Experiment 1, however, no Program or Flow Aid was given and half the subjects in each instructional group received only Interpretation exercises in Practice Sets 1 and 2 (Interpretation Practice) and half received only Generation items (Generation Practice). All subjects received both kinds of problems in the posttest, as in Experiment 1.

Method

Subjects and design. The subjects were 56 Indiana University students who participated in the experiment in order to fulfill a requirement for their introductory psychology course. Each subject served in one cell of a 2x2x2 design with the first factor being instructional text (Model vs. Rule Text), the second factor being type exercises given in Set 1 and 2 (Interpretation vs. Generation Practice), and the third factor being mathematical ability as measured by MSAT score (High vs. Low Ability). Since all subjects received a posttest consisting of both types of exercises, comparisons by type of problem (Generation vs. Interpretation Exercises) were within subjects comparisons.

Materials. The Model Text and Rule Text, the three sets of exercises, the answer sheets, the Pre-Experimental Questionnaire, and the Algebra Pre-Test were all identical to or slightly revised from those used in Experiment 1.

Procedure. As in Experiment 1, subjects were run in small groups of 2 to 4 and began by completing the Pre-Experimental Questionnaire and Algebra Pre-Test. Following instructions, each subject was given his instructional

text -- either Model or Rule Text -- and read it as in Experiment 1. The "program aid" and "flow aid" were not included. Then each subject was given Exercise Set 1 followed by Set 2, as in Experiment 1, except that all items in the first two sets were of same type (either all Generation or all Interpretation Practice) and the subject continued on a set to a criterion of 6 out of 8 correct (rather than 4 in a row). After reaching criterion on the first two sets of exercises, the subject was given the third set (posttest) as in Experiment 1 with 5 Generation items and 5 interpretation items arranged in alternating order.

Results

Two subjects indicated previous experience with computer programming, 3 subjects were unable to solve 3 out of 6 problems on the algebra pretest, and 4 subjects were unable to reach criterion on Set 1 or Set 2 within 2 hours. The data for these subjects were eliminated and new subjects were run in their places. Subjects indicating MSAT scores of 560 or above were counted as High Ability and those with scores below 560 were counted as Low Ability. The performance on the exercises was scored as in Experiment 1.

The proportion correct response on the posttest for the two instructional groups by type of practice experience (on Sets 1 and 2) and by mathematical ability is given in Table 4. As can be seen, and as is indicated by an ANOVA, the Model x Type of Practice interaction is reliable ($F(1,48) = 6.50$, $p < .025$), indicating that for Model subjects practice on generating programs helped most on increasing performance on the posttest, but for Rule subjects practice on interpretation of programs helped most. In this study, practice seems to have the effects of "filling in" on material not emphasized in the text.

Table 4

Proportion Correct Response on Transfer Posttest for Four Instructional x Practice Groups

Instructional Group	Type of Practice		
	Generation	Interpretation	
High Ability			
Rule	.67	.60	.64
Model	.59	.44	.51
Low Ability			
Rule	.34	.56	.45
Model	.64	.36	.50

Note. - Main effect of Model, p = n.s.; main effect of type of practice, p = n.s.; Model x Practice interaction, $p < .025$; Model x Ability interaction, $p < .15$; Model x Practice x Ability interaction, $p < .10$.

There were two marginally reliable effects involving ability. The Model x Ability interaction ($F(1,48) = 2.34$, $p < .15$) showed the same general pattern as in Experiment 1 with Model Text resulting in higher proportion correct response than Rule Text for Low Ability subjects (55% vs. 45% correct, respectively) and the reverse true for High Ability subjects (51% vs. 64% correct, respectively). In addition, the marginally reliable Model x Type of Practice x Ability interaction shown in Table 4 ($F(1,48) = 3.66$, $p < .10$) suggests the Model x Type of Practice interaction is much more pronounced for Low Ability subjects than for High Ability subjects. These findings are consistent with the idea that the model text provides subjects with a meaningful model which is especially important for Low ability subjects, but which may interfere with High ability subjects.

Table 4 about here

Unlike Experiment 1, there was no reliable Model x Type of Problem Interaction ($F < 1.00$); however, an investigation of subjects receiving generation practice -- a sort of neutral exercise -- did provide a hint of the interaction with Model subjects excelling on Interpretation items and Non-Model subjects excelling on Generation items.

GENERAL DISCUSSION

Availability of learning set. These results provide important new information concerning the conditions of meaningful learning of technical material. Ausubel's distinction between meaningful and rote learning set seems to be exemplified by differences between Model and Non-Model treatments. Model

instruction provides the learner with a rich set of prior experiences which are familiar to the learner and by which new information may be understood and organized; since the model is presented first and new material is then related to it, it shares some of the characteristics of Ausubel's "advance organizer".

The results allow some understanding of what makes a good advance organizer. The program aid (Experiment 1) and the flow chart (Experiment 2) did not seem to serve as useful advance organizers for subjects in our experiments, i.e., they did not provide a meaningful assimilative set. The program aid was not familiar to learners and although it provided "organization" for the seven statements, it did not provide subjects with a means of tying new information to existing knowledge. The flow chart aid presented geometric symbols which were apparently familiar to subjects, but the symbols themselves provided only a second layer of code (i.e., translating statements to arbitrary symbols) rather than an organizing superstructure. The model aid, on the other hand, provided a superstructure already familiar to learners and to which new information could be systematically related; non-model subjects including those given the program or flow chart aids apparently had to use a rote learning set which lacked a rich set of relevant experience.

In previous studies (Mayer & Greeno, 1972; Mayer, 1974) we have noted two structural variables in the acquisition of new knowledge. External connections refer to links between new material and a system of knowledge already in a learner's cognitive structure. This rich set of experiences is what Ausubel calls a meaningful learning set or Greeno (1972) terms "semantic memory". For example, understanding of the relation between counter set statements and the

memory scoreboard is an example of external connection. Internal connections refers to links between one aspect of new material and another aspect of new material which retains the original structure of the material. These kinds of links may be acquired when a learner lacks a rich set of relevant experience, i.e., what Ausubel calls rote learning set or what Greeno terms "algorithmic knowledge". An example is knowing that in counter set statements the memory space is always on the left of the equals sign and the number is always on the right is an example of internal connection.

In the present experiments it seems reasonable to propose that Model subjects had a meaningful learning set active during learning and so acquired cognitive structure with strong external connections but weak internal connections; on the other hand, Non-Model subjects used only a rote learning set of experience with arithmetic and technical systems and so acquired cognitive structure with strong internal and weak external connections.

This interpretation is consistent with the results of Experiments 1 and 2 in which Model subjects excelled on learning and on transfer to problems requiring interpretation and extension of presented material, while Non-Model subjects excelled on straightforward generation of programs similar to those in the booklet. The different patterns of posttest performance for Model and Non-Model groups is reminiscent of earlier results with mathematics learning (Mayer & Greeno, 1972; Mayer, 1974) and suggests that the two groups acquired learning outcomes which differed in qualitative or structural ways. These results provide an important extension of earlier findings because they deal with a new type of subject matter (technical instruction for computer programming) and because they more clearly demonstrate that an important variable in

instruction -- in addition to the presentation of needed facts -- is the presence or absence of a conceptual model. These results support the idea that instruction for technical information can be made "meaningful" for novices, and that the effects of meaningful learning can be assessed in terms of structurally different learning outcomes.

Effects of question answering activity. The answering of practice exercises was varied in Experiments 2 and 3. In Experiment 2, varying the amount of practice before the posttest had no observable effect on posttest performance, although this may be due to the fact that differences in the amount of practice were not large. However, in Experiment 3 varying the type of practice before the posttest -- i.e., practice on generation vs. practice on interpretation -- had interesting effects. Generation practice increased performance most for the Model groups and interpretation practice increased performance most for the Non-Model subjects (especially on interpretive items). These results are consistent with the notion that practice may serve to direct the learner's attention to aspects of material not emphasized in instruction, especially helping the Non-Model subjects to work on extending and interpreting presented material. One suggestion is that, during practice, subjects learn to "fill-in" abilities not acquired in instruction and thus eliminate differences in "what is learned" by activating complimentary learning sets. In the present study, however, only the "backwards effects" of practice -- i.e., as a review -- were investigated since no text ever followed practice; further work should investigate both forward and backwards effects as suggested by Frase (1968) and McGraw & Grotelueschen (1972).

Effect of learner aptitude. Subjects with prior experience in mathematics, and other areas related to computer programming, may possess a meaningful learning set independent of the model presented in instruction. Since the model may be a rather arbitrary and contrived crutch for learners, it may actually interfere with high ability learners who already have a rich set of more sophisticated knowledge, while at the same time providing a meaningful learning set to learners low in ability.

Only weak support for this idea was provided in the present experiments. Slight or marginally reliable interactions involving aptitude and treatment (ATI) were obtained in Experiments 1 and 3 with Model instruction raising the scores of low math ability subjects most and Non-Model instructions helping raise posttest scores of high ability learners most. However, Experiment 2, which investigated five types of abilities failed to yield any reliable interactions involving ability. Thus, there is only weak support for the idea that high mathematics ability may function in the same way as an experimentally induced meaningful learning set, and further work is required in this area.

The analysis of tests in Experiment 2 did, however, yield an interesting possibility that learning problems can be predicted by appropriate pretests and instruction emphasized in these areas. Further work should investigate the role of specifically relevant skills and of traditional computer programming aptitude tests such as those used in the selection of computer programmers (Luftig, 1973).

References

Ausubel, D.P. Educational Psychology: A Cognitive Approach. New York: Holt, Rinehart & Winston, 1968.

Bracht, G.H. Experimental factors related to aptitude-treatment interaction. Review of Educational Research, 1970, 40, 627-645.

Brownell, W.A. Psychological considerations in the learning and teaching of arithmetic. In: The Teaching of Arithmetic: Tenth Yearbook of the National Council of Teachers of Mathematics. New York: Bureau of Publications, Teachers College, Columbia University, 1935, pp. 1-35.

Cronback, L.J. and Snow, R.E. Individual differences in learning ability as a function of instructional variables. Final Rep. Sch. Educ., Stanford University, Contrast No. OEC-4-6-061269-1217, USOE, 1969.

Frase, L.T. Some data concerning the mathemagenic hypothesis. American Educational Research Journal, 1968, 5, 181-189.

Gould, J. Querying by nonprogrammers. Paper presented at annual meeting of American Psychological Association, 1974.

Greeno, J.G. The structure of memory and the process of solving problems. University of Michigan: Human Performance Center Technical Report No. 37, 1972.

Kreitzberg, G.B. and Swanson, L. A cognitive model for structuring an introductory programming curriculum. Proceedings of the National Computer Conference, 1974, Pp. 307-311.

Katona, G. Organizing and Memorizing. New York: Columbia University Press, 1940.

Leskow, S. and Smock, C.D. Developmental changes in problem solving strategies: permutations. Developmental Psychology, 1970, 2, 412-422.

Luftig, M. Computer Programmer Aptitude Tests, New York: ARCO, 1972.

Mayer, R.E. Dimensions of learning to solve problems. University of Michigan: Human Performance Center Memorandum Report No. 14, 1972.

Mayer, R.E. and Greeno, J.G. Structural differences between learning outcomes produced by different instructional methods. Journal of Educational Psychology, 1972, 63, 165-172.

Mayer, R.E. Acquisition and resilience under varying testing conditions of structurally different problem solving procedures. Journal of Educational Psychology, 1974, 65, 644-656.

McGraw, B. & Grotelueschen, A. Direction of the effect of questions on prose material. Journal of Educational Psychology, 1972, 63, 580-588.

Miller, L. Computer programming by nonprogrammers. IBM Technical Report, 1972.

Papert, S. A computer laboratory for elementary schools. M.I.T. Artificial Intelligence Laboratory LOGO Memo No. 1, 1971.

Piaget, J. Science of Education and the Psychology of the Child. New York: Orion Press, 1970.

Rothkopf, E.Z. The concept of mathemagenic activities. Review of Educational Research, 1970, 40, 325-336.

Roughead, W.G. and Scandura, J.M. What is learned in mathematical discovery. Journal of Educational Psychology, 1968, 59, 283-289.

Shneiderman, B. and Ho, C. Experiment study of computer programming by novices and experienced programmers. Indiana University: Department of Computer Science Technical Report No. 17, 1974.

Sime, M.E., Green, T.R.G. and Guest, D.J. Psychological evaluation of two conditional constructions used in computer languages. International Journal of Man Machine Studies, 1973, 5, 105-113.

Weinberg, G.M. The Psychology of Computer Programming. New York: Van Nostrand, 1971.

Weissman, L.M. A methodology for studying the psychological complexity of computer programs. University of Toronto: Computer Systems Research Group Technical Report No. 37, 1974.

Footnotes

1. This research was sponsored by Grant EC-44020 from the National Science Foundation. The author also wishes to acknowledge Christine Ward-Hull and Don Young who assisted in the collection of data. Finally, the author wishes to thank Frank Restle for his many useful comments and encouragements.

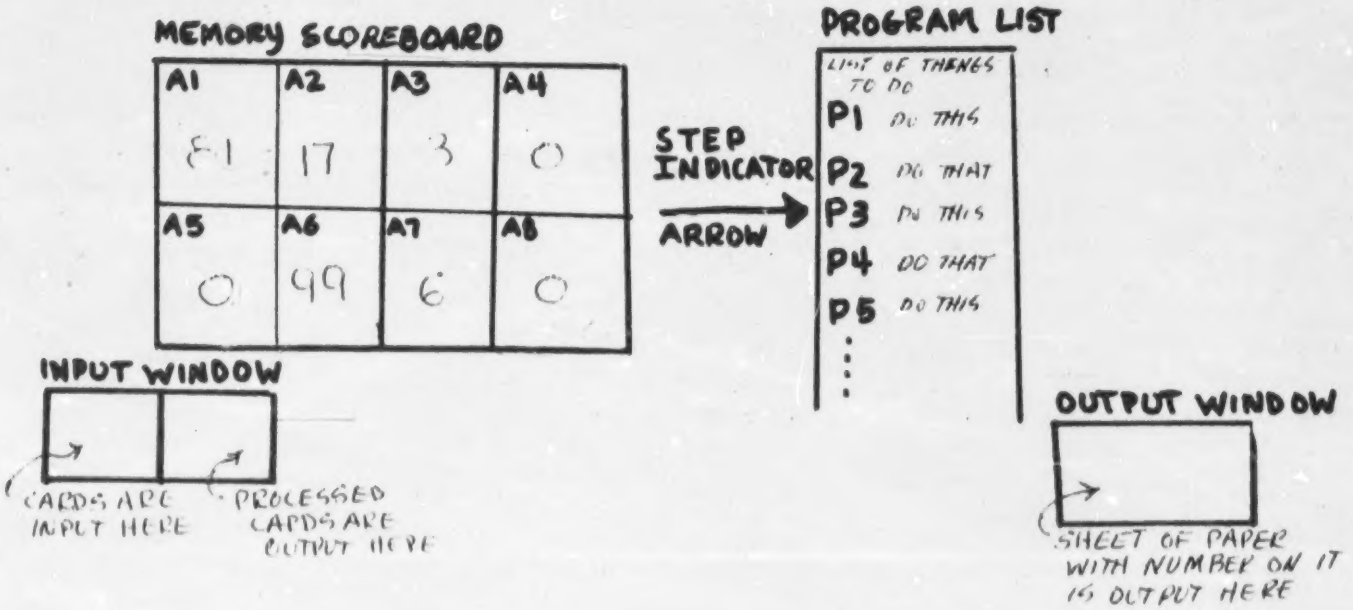
2. Requests for reprints should be sent to author at: Department of Psychology, Indiana University, Bloomington, Indiana 47401.

APPENDIX A

Text of Model and Rule Booklets

Model Booklet:

How Does a Computer Operate?



The figure above represents a simple computer system which will be explained to you in this booklet. It is made up of three main parts: (1) INPUT & OUTPUT which allows communication between the computer's memory and the outside world, (2) MEMORY which stores information in the computer, and (3) PROGRAM which tells the computer what to do and what order to go in. Each of these three parts will now be explained.

INPUT & OUTPUT: Notice that to the far left is an input window divided into two parts. A pile of computer cards with numbers punched into them can be put in the left part of the window; as the computer finishes processing each card it puts it on the right side of the input window. Thus when the computer needs to find the next data card, it takes the top card in the left side of the input window; when it is done with the card, it puts it on the right side.

On the far right is the output window. This is where printed messages (in this case, only numbers can be printed) from the computer's memory to the outside world appear. Each line on the printout is a new message (i.e., a new number).

Thus the computer can store a number in memory that is on a card at the input window or it can print out what it has in memory onto a printout at the output window. The statements which put the input and output windows to work are READ and WRITE statements, and each will be explained later on.

MEMORY: Inside the computer is a large scoreboard called MEMORY. Notice that it is divided into eight spaces with room for one score (i.e., one number) in each space. Also notice that each space is labeled with a name -- A1, A2, A3, A4, A5, A6, A7, A8. These labels or names for each space are called "addresses" and each of the eight addresses always has some number (score) indicated in its space. For example, right now in our figure A1 shows a score of 81, A2 shows a score of 17, etc.

It is possible to change the score in any of the eight spaces; for example, the score in box A1 can be changed to 0, and you will learn how to change scores in memory later on when we discuss EQUALS statements and ARITHMETIC statements.

PROGRAM: Inside the computer to the right of the MEMORY scoreboard (with its eight address-score pairs), is a place to put a list of things to do called PROGRAM and an arrow which indicates what step in the list the computer should work on.

Notice that each line in PROGRAM has a number with the first line called P1, the second step called P2, and so on. When a program is inserted the step indicator arrow will point to the first line (P1); when the first step is finished the arrow will go to the next step on the list (P2), and so on down the list. The pointer arrow will follow this procedure of pointing to the steps in order, from the top down, unless it comes to a step which tells it to point to some other step -- then it will go to that step and accomplish it, and start working down the list from there. This is called "looping" because the arrow is not going in a straight line. For example, the pointer may first point to P1, then the computer will finish step P1, then the pointer will shift to P2 and the computer will finish step P2, then the pointer will shift to P3 but P3 may say to go to step P7 in which case the pointer will shift to P7 (skipping P4, P5 and P6) and the computer will do P7, then P8, and so on. You will learn how to control the order of steps in the program later on when the IF statement and GO TO statement and STOP statements are discussed.

READ and WRITE Statements

First the statements that have to do with INPUT and OUTPUT will be presented.

The input statement duplicates a number which is on a data card into a memory space (i.e., a scoreboard box) and is in the form,

READ ( )

where an address name goes inside the parentheses. Remember that an address name is just a space in the memory scoreboard, and in this experiment we will assume there are eight memory spaces called A1, A2, A3, A4, A5, A6, A7, A8.

For example, the statement

READ (A2)

means that BEFORE this statement there is a pile of cards (i.e., at least one) waiting in the left side of the input window and some unknown number is being stored in memory scoreboard space A2, but AFTER this statement is finished the top data card has moved to the right half of the input window and the number which was punched in this card is now stored on the scoreboard at space A2 (instead of whatever score was there before).

In terms of what operations are performed, the statement

READ (A2)

means: (Assume a pile of data cards has been put on the left side of the input window.) (1) Take the top card from the pile of cards in the input window and check to see the first number punched into it. (2) Store that number in a place on the memory scoreboard called A2, destroying any previous number which was stored at A2. (3) Send that data card to the right side of the input window reducing the pile of cards to-be-processed by one.

The output statement, on the other hand, duplicates a number that is on the memory scoreboard onto a printout at the output window, and is in the form,

WRITE ( )

where an address name goes in the parentheses. Remember that an address name is just a space on the memory scoreboard and that there are eight of them: A1, A2, A3, A4, A5, A6, A7, A8.

For example, the statement

WRITE (A1)

means that BEFORE this statement there is a number stored on the memory scoreboard at box A1, but after this statement is finished the number is still on the scoreboard as before and it is also printed out on one line of a piece of paper at the output window.

In terms of the operations performed, the statement

WRITE (A1)

means: (1) Check the scoreboard to see what number is in box A1, but do not alter it. (2) Print out that number on a piece of paper and send it out the output window.

EQUALS and ARITHMETIC Statements

Now, that you have some idea how statements effect INPUT & OUTPUT, you will learn about two program statements which effect MEMORY especially changes in the scores on the memory scoreboard.

One kind of statement that can change the number stored in memory (without READing a card) is expressed in the form,

_____=_____

where the first blank is an address name (i.e., a box on the scoreboard) and the blank to the right of the equals sign is either another address name or a number.

For example, the statement

A1=0

means that BEFORE this statement some unknown number is being stored on the scoreboard in space A1 but AFTER this statement is finished the number zero is being stored in space A1 (instead of whatever other score was there before).

In terms of what operations are performed, the statement

A1=0

means: (1) Destroy whatever was previously stored on the scoreboard at memory space A1. (2) Store the number zero in memory space A1.

Another example is the statement

A4=A5

which means that BEFORE this statement some number is stored on the scoreboard in memory space A4 and some number is stored on A5, but AFTER this statement is finished the original number is showing at A5 and it is also now showing at box A4 (instead of whatever was there before). In terms of what operations are performed this statements means: (1) Destroy whatever number was stored at memory space A4. (2) See the number which is stored in memory space A5, and without altering it, store it also in memory space A4.

Another statement that changes the memory scoreboard (without READing in a card) is just like the EQUALS statement except that a computation is indicated on the right of the equals sign. The statement is

_____=_____ __ _____

where the first blank (on the left of the equals) is an address name, the first blank on the right of the equals is either a number or an address name, the second blank on the right of the equals is an operation (addition, subtraction, multiplication, division) and the third blank on the right is either a number or an address name. The four arithmetic operations are expressed as follows: + means add, - means subtract, * means multiply, and / means divide.

For example, the statement

A1=A1+A2

means that BEFORE the statement some number is on the scoreboard in memory box A1 and some number is also on the board in box A2, but AFTER the statement is finished the original number is still in box A2 but the number now on the scoreboard in space A1 is the sum of the original number in A1 and the number in A2.

In terms of the operations performed, the statement

A1=A1+A2

means: (1) Check to see what number is on the scoreboard at box A1 and what number is in box A2 but do not alter these scores. (2) Add these two numbers together. (3) Take down and destroy the number that was on the board at A1 and put this new sum in box A1 instead.

GO TO, IF and STOP Statements

Now that you have some idea about statements that effect INPUT & OUTPUT, and statements that effect changing MEMORY, you will learn about statements which effect the PROGRAM part of the computer -- the order in which statements will be carried out.

Normally when a list of statements is put together into a program and inserted into the PROGRAM part of the computer, the arrow will point to the first statement in the list, wait for it to be carried out, then point to the second and so on down the list. However, it is possible to tell the pointer arrow to point to some other statement rather than the one directly below it.

An example is the statement,

GO TO _____

where a statement number goes in the blank. Remember that each statement in a program has its own line and is given a number such that the first statement is called P1, the second statement is P2, the third is P3 and so on; the statement number goes in the margin just to the left of the statement.

For example, the statement

GO TO P2

means that BEFORE the statement the arrow was pointing to and the computer was finishing work on the statement above "GO TO P2" in the program, and AFTER the statement is completed the arrow is pointing to the statement at P2 -- i.e., the second one in the list -- and the computer will begin working on it.

In terms of the operations involved, the statement

GO TO P2

means: (Assume the computer has just finished the statement before "GO TO P2" in the PROGRAM.) (1) Do not point the arrow to nor work on the statement that comes right after "GO TO P3" in the program as normally would be done. (2) Instead, shift the arrow to the second statement in the PROGRAM list and start working on it. (And then go on down the line to P3 and so on from there.)

The last kind of statement that controls where the arrow points on the PROGRAM list is one that comes at the end of the list and means work on the program is over. This is the statement,

STOP.

For example, the statement

STOP

means that BEFORE the statement the computer has just finished the statement above "STOP" (or has been directed to the STOP statement by a GO TO statement), and AFTER the statement has been finished the computer is finished with the present PROGRAM list and ready for a new one, i.e., the arrow is finished pointing to statements.

In terms of operations, the statement

STOP

means: (1) You've come to the end of things to do with this program so keep the arrow still and stop working on it. (2) Start looking for a new PROGRAM list to work on.

Remember that the computer will work on whatever statement the arrow points to on PROGRAM and that once the computer finishes one statement, the arrow will shift to the next statement in line unless it comes to a statement that tells the arrow to "GO TO" some other statement. It is also possible to tell the arrow to "GO TO" some other statement under certain conditions and to go to the next statement in line under other conditions.

This is done by using the statement,

IF(_____ __ _____) GO TO _____

where the first blank in the parentheses after IF is an address name, the middle blank is a relationship (less than, less than or equal, equal, greater than or equal, greater than), the third blank is a number, and the blank after GO TO is a statement number. The symbols for the five relationships are: < means less than, ≤ means less than or equal, = means equal, ≥ means greater than or equal, > means greater than.

For example, the statement

IF(A2=99) GO TO P7

means that BEFORE this statement there is a number stored in memory box A2 and the arrow was pointing at the statement just above this one in PROGRAM, and AFTER this statement is finished the arrow will be pointing to the seventh statement in the program (P7) if the number stored in memory space A2 is 99 or it will be pointing to the statement just below this IF statement if the number in A2 is not 99.

In terms of the operations involved, the statement

IF(A2=99) GO TO P7

means: (Assume the arrow has just finished pointing to the statement above this one.) (1) Check to see what number is stored in memory box A2 but do not change it. (2) If the number is not 99, just shift the arrow normally to the statement that comes after the IF statement in the program. (3) If, however, the number is 99 shift the arrow to the seventh statement in the program (ignoring all others inbetween).

Another example is the statement

IF(A2≥5) GO TO P2

which means that BEFORE this statement there is a number stored at memory space A2 and the computer has just finished working on the statement before the IF statement in the program, but AFTER this statement is finished the computer will be working on the second statement in the PROGRAM list (P2) if the number stored in A2 is greater than or equal to 5 or it will be working on the statement just below this IF statement if the number is not. In terms of operations this statement means: (1) Check to see what number is stored in memory space A2 but do not alter it. (2) If the number is not greater than or equal to 5, continue normally with the statement that comes just after the IF statement in the program. (3) However, if the number is greater than or equal to 5, start working on the second statement in the program.

Rule Booklet:

What is a Computer Language?

In this booklet you will learn how to write seven different kinds of computer statements, and what they mean in ordinary English. You can think of each statement as a kind of sentence written in computer language, which tells the computer to perform certain operations. Statements can be put together in various ways into lists, with one statement per line; these lists are called programs because they tell the computer to perform a whole series of tasks in a certain order. In order to write programs, you must first learn about each of seven statements that can be put into a program.

Each of the seven statements will now be explained.

READ Statements

The first kind of statement is

READ ( )

where an "address" name goes inside the parentheses. An address name is just a space in the computer's memory, and in this experiment there are eight memory spaces called A1, A2, A3, A4, A5, A6, A7, A8.

For example, the statement

READ (A2)

means: (Assume a pile of data cards has been input to the computer.) (1) Take the top card from the pile of cards input to the computer and check to see the first number punched into it. (2) Store that number in a place in memory called A2, destroying any previous number which was stored at A2. (3) Send that data card out of the computer reducing the size of the pile of cards by one.

WRITE Statements

The next statement you will learn about is

WRITE ( )

where an address name goes in the parentheses. Remember that an address name is just a space in memory and that there are eight of them in our computer: A1, A2, A3, A4, A5, A6, A7, A8.

For example, the statement

WRITE (A1)

means that BEFORE this statement there is a number stored in memory space A1, but AFTER this statement is finished the number is still in memory space A1 and is printed out on one line of a piece of paper which is output.

In terms of the operations performed, the statement

WRITE (A1)

means: (1) Check to see what number is stored in memory at space A1, leaving that number unchanged. (2) Print out that number on a piece of paper and send it out the output window.

EQUALS Statements

The next kind of statement you will learn about is expressed in the form,

_____=_____

where the first blank is an address label and the second blank (i.e., the blank to the right of the equals sign) is either a number or another address name.

For example, the statement

A1=0

means that BEFORE this statement some unknown number is being stored in space A1 but AFTER this statement is finished the number zero is being stored in space A1 (instead of whatever was there before).

In terms of what operations are performed, the statement

A1=0

means: (1) Destroy whatever number was previously stored at memory space A1. (2) Store the number zero in memory space A1.

Another example is the statement

A4=A5

which means that BEFORE this statement some number is stored in memory space A4 and some number is stored at A5, but AFTER this statement is finished the original number is still stored at A5 and it is now also stored at A4 (instead of what was there before). In terms of what operations are performed this statement means: (1) Destroy whatever number was stored at memory space A4. (2) See the number which is stored in memory space A5 and without destroying it, store it also in memory space A4.

ARITHMETIC Statements

The next kind of statement is just like the EQUALS statement except that a computation is indicated on the right side of the equals sign. The statement is

_____=_____ __ _____

where the blank on the left of the equals sign is an address name, the first blank on the right of the equals is either a number or an address name, the second blank on the right is an operation (either addition, subtraction, multiplication, division) and the third blank on the right is either a number or an address label. The four arithmetic operations are expressed as follows: + means add, - means subtract, * means multiply, and / means divide.

For example, the statement

A1=A1+A2

means that BEFORE the statement some number is stored in memory space A1 and some number is stored in memory space A2, but AFTER the statement is finished the original number if still in A2 but the number now stored in A1 is the sum of the original number in A2 plus the number in A2.

In terms of the operations performed, the statement

A1=A1+A2

means: (1) Check to see what number is stored at memory space A1 and what number is stored at memory space A2 but do not alter them. (2) Add these two numbers together. (3) Now destroy whatever number was stored at A1 and put this new sum in A1 instead.

GO TO Statements

Normally when a list of statements is put together into a program, the computer will complete the first statement, then go on to the one directly below it and so on down the list. However, it is possible to tell the computer to go to some other statement rather than the one directly below it.

An example is the statement

GO TO __

where a statement number goes in the blank. Each statement in a program has its own line and it is given a number such that the first statement is called P1, the second statement is called P2, the third is P3 and so on; the statement number goes in the margin just to the left of the statement.

For example, the statement

GO TO P2

means that BEFORE the statement the computer has finished the statement just above "GO TO P2" on the program, and AFTER the statement is completed the computer will begin working on whatever statement is the second one in the program (i.e., statement P2).

In terms of the operations involved, the statement

GO TO P2

means: (Assume the computer has just finished the statement before "GO TO P2".) (1) Do not work on the statement that comes just after "GO TO P2" in the program as normally would be done. (2) Instead work on the statement numbered P2 -- the second statement in the program (and then go to P3 and so on).

IF Statements

Remember that once a computer finishes one statement it will go on to work on the statement that comes directly after (i.e., below) it on the program, unless that statement tells the computer to "GO TO" some other statement. It is also possible to tell the computer to "GO TO" some other statement number under certain circumstances and to go on to the next statement in line under other circumstances.

This is done by using the statement

IF(_____ __ _____) GO TO_____

where the first blank in the parentheses after the IF is an address name, the middle blank is a relationship (either less than, less than or equal, equal, greater than or equal, greater than), the third blank in the parentheses is a number, and the blank after GO TO is a statement number. The symbols for the five relations are: $<$ means less than, $\leq$ means less than or equal, $=$ means equal, $>$ means greater than, and $\geq$ means greater than or equal.

For example, the statement

IF(A2=99) GO TO P7

means that BEFORE this statement there is a number stored at memory space A2 and the statement just above this one in the program has been completed, and AFTER this statement is completed the computer will be working on the seventh statement in the program if the number stored in memory space A2 is 99 or it will be working on the statement just below this IF statement if the number in A2 is not 99.

In terms of the operations involved, the statement

IF (A2=99) GO TO P7

means: (Assume the computer has just finished the statement above this one in the program.) (1) Check to see what number is stored in memory space A2 but do not change it. (2) If the number is not 99, just continue normally with the statement that comes after the IF statement in the program. (3) However, if the number is 99, start working on the seventh statement (ignoring all others in between).

Another example is the statement

IF(A2≥5) GO TO P2

which means that BEFORE this statement there is a number stored at memory space A2 and the computer has just finished the statement directly before

the IF statement in the program, but AFTER this statement is finished the computer will be working on the second statement in the program if the number stored in space A2 is greater than or equal to 5 or it will be working on the statement just below this IF statement if the number is less than 5. In terms of operations this statement means: (1) Check to see what number is stored in memory space A2 but do not alter it. (2) If the number is not greater than or equal to 5, continue normally with the statement that comes just after the IF statement. (3) However, if the number is greater than or equal to 5, start working on the second statement in the program.

STOP Statements

The last kind of statement that you will learn about today is one that comes at the end of a program and means the program is over. This is

STOP.

For example, the statement

STOP

means that BEFORE the statement the computer has just finished the statement above it (or has been directed to the STOP statement) by a GO TO statement), and AFTER the statement has been finished the computer is finished with the entire program and ready for a new one.

In terms of operations, the statement

STOP

means: (1) You've come to the end of things to do with this program so stop working on it. (2) Start looking for some other program that needs to be started.

APPENDIX B

Typical Practice and Test Questions

Generation-Statement:

1. Given the computer has just finished statement P2 in a program and a number is in memory space A8, write a statement to get the computer to statement P7 if the number is equal to 8 and to statement P4 if it is not.

2. Given a data card with a number on it is input, write a statement to get that number into memory space A1.

3. Given a number in memory space A5, write a statement to change that number to zero.

4. Given a number in memory space A6, write a statement to increase that number by 1.

5. Given a computer has just finished statement P7 on a program, write a statement to get the computer immediately to the third statement (P3).

6. Given a number in memory space A1 and a number in memory space A2, write a statement to find their product and store that number in space A2.

7. Given a number in memory space A3, write a statement to change that number to 5.

8. Given a data card with a number on it is input, write a statement to get that number into memory space A5.

9. Given a number in memory space A7, write a program to decrease that number by 1.

10. Given the computer has just finished statement P7 in a program and a number is in memory space A1, write a statement to get the computer to statement P2 if the number is equal to 5 and to statement P9 if it is not.

11. Given the computer has just finished statement P3 on a program, write a statement to get the computer immediately to the seventh statement (P7).

12. Given a number in memory space A1 and a number in memory space A2, write a statement to find the sum of those two numbers and store the sum in A1.

Interpretation-Statement:

1. P3 IF(A8=8) GO TO P7
2. READ (A1)
3. A5=0
4. A6=A6+1
5. P8 GO TO P3
6. A2=A1*A2
7. A3=5
8. READ (A5)
9. A7=A7-1
10. P8 IF(A1=5) GO TO P2
11. P4 GO TO P7
12. A1=A1+A2

Generation-Linear Program:

1. Given a card with a number on it is input, write a program to print out its square.
2. Given two cards with a number on each are input, write a program to find their sum.
3. Given a card with a number on it is input, write a program to print out that number unless it is zero.
4. Given that a number is stored in memory space A2 and that a card with a number on it is input, write a program to find the product of the two numbers.
5. Given a number is stored in memory space A6, write a program to print out that number unless it is 99.
6. Given a card with a number on it is input, write a program to print out that number minus one.
7. Given two cards with a number on each are input, write a program to find their product.

8. Given a card with a number on it is input, write a program to print out twice that number.

9. Given that a number is stored in memory space A1 and that a card with a number on it is input, write a program to find the sum of the two numbers.

10. Given a card with a number on it is input, write a program to print out half that number.

11. Given a card with a number on it is input, write a program to print out that number unless it is less than 2.

12. Given a number is stored in memory space A1, write a program to print out the number unless it is greater than 5.

Interpretation-Linear Program

1.
```
P1  READ (A1)
P2  A1=A1*A1
P3  WRITE (A1)
P4  STOP
```

2.
```
P1  READ (A1)
P2  READ (A2)
P3  A1=A1+A2
P4  WRITE (A1)
P5  STOP
```

3.
```
P1  READ (A1)
P2  IF(A1=0) GO TO P4
P3  WRITE (A1)
P4  STOP
```

4.
```
P1  READ (A1)
P2  A1=A1*A2
P3  WRITE (A1)
P4  STOP
```

5.
```
P1  IF (A6=99) GO TO P3
P2  WRITE (A6)
P3  STOP
```

6.
```
P1  READ (A1)
P2  A1=A1-1
P3  WRITE (A1)
P4  STOP
```

7.
```
P1  READ (A1)
P2  READ (A2)
P3  A1=A1*A2
P4  WRITE (A1)
P5  STOP
```

8.
```
P1  READ (A1)
P2  A1=A1*2
P3  WRITE (A1)
P4  STOP
```

9.
```
P1  READ (A2)
P2  A1=A1+A2
P3  WRITE (A1)
P4  STOP
```

10.
```
P1  READ (A1)
P2  A1=A1/2
P3  WRITE (A1)
P4  STOP
```

11.
```
P1  READ (A1)
P2  IF (A1<2) GO TO P4
P3  WRITE (A1)
P4  STOP
```

12.
```
P1  IF (A1>5) GO TO P3
P2  WRITE (A1)
P3  STOP
```

Generation-Looping Program:

1. Given that a pile of data cards is input, write a program to print out only numbers greater than 5 and to stop when it gets to a number greater than 50.

2. Given a pile of cards is input, write a program to count (and store in memory) how many cards there are before a card with a 10 appears.

3. Given a pile of data cards with a number on each is input, write a program to print out the double of each number and to stop when it gets to a card with a 0 on it.

4. Given a pile of data cards is input, write a program to print out each number and stop when it gets to a card with 88 on it.

5. Given a pile of data cards is input and that a number is stored in memory space A6, write a program to print out the difference between each number and A6 and to stop when it gets to a card with 99 on it.

6. Given a pile of cards is input, write a program to count (and store in memory) how many cards there are before a card with a 99 appears.

7. Given a pile of data cards is input, write a program to print out only numbers greater than 8 and to stop when it gets to a number less than or equal to 2.

8. Given a pile of data cards with a number on each is input, write a program to print out the square of each number and stop when it gets to a card with 99 on it.

9. Given that a pile of data cards is input and a number is stored in memory space A2, write a program to print out each number plus whatever is in A2 and to stop when it gets to a card with 77 on it.

10. Given a pile of data cards is input, write a program to print out half of each number and stop when it gets to a card with 77 on it.

Interpretation-Looping Programs

1.
```
P1  READ (A1)
P2  IF (A1>50) GO TO P6
P3  IF (A1≤5) GO TO P1
P4  WRITE (A1)
P5  GO TO P1
P6  STOP
```

2.
```
P1  A1=0
P2  READ (A2)
P3  IF (A2=10) GO TO P6
P4  A1=A1+1
P5  GO TO P2
P6  STOP
```

3.
```
P1  READ (A1)
P2  IF (A1=0) GO TO P6
P3  A1=A1*2
P4  WRITE (A1)
P5  GO TO P1
P6  STOP
```

4.
```
P1  READ (A1)
P2  IF (A1=88) GO TO P5
P3  WRITE (A1)
P4  GO TO P1
P5  STOP
```

5.
```
P1  READ (A1)
P2  IF (a1=99) GO TO P6
P3  A1=A6-A1
P4  WRITE (A1)
P5  GO TO P1
P6  STOP
```

```
6.  P1  A1=0
    P2  READ (A2)
    P3  IF (A2=99) GO TO P6
    P4  A1=A1+1
    P5  GO TO P2
    P6  STOP

7.  P1  READ (A1)
    P2  IF (A1≤2) GO TO P5
    P3  IF (A1≤8) GO TO P1
    P4  WRITE (A1)
    P5  GO TO P1
    P6  STOP

8.  P1  READ (A1)
    P2  IF (A1=99) GO TO P6
    P3  A1=A1*A1
    P4  WRITE (A1)
    P5  GO TO P1
    P6  STOP

9.  P1  READ (A1)
    P2  IF (A1=77) GO TO P6
    P3  A1=A1+A2
    P4  WRITE (A1)
    P5  GO TO P1
    P6  STOP

10. P1  READ (A1)
    P2  IF (A1=77) GO TO P6
    P3  A1=A1/2
    P4  WRITE (A1)
    P5  GO TO P1
    P6  STOP
```

APPENDIX C

Four Pre-tests Used in Experiment II

Pre-test 1: Algebra Solution Test

1. $Y = X + 5$, $X = 5$, $Y =$ _____

2. $2Y = X$, $X = 10$, $Y =$ _____

3. $2Y + 3 = 7$, $Y =$ _____

4. $Y^2 - 6 = 10$, $Y =$ _____

5. $X = Y + 2C$, $Y = A - B$, $X =$ _____ (in terms of A, B and/or C)

6. $x = 6Z - Y$, $Z = B - 1$, $Y = A$, $X =$ _____ (in terms of A, B and/or C)

Pre-test 2: Algebra Story Test

1. A taxicab charges 25¢ for the first fifth of a mile and five cents for each additional fifth of a mile. Find the formula for the cost C in dollars of going M miles by taxi. Assume M is greater than 1/5.

(1)_____

2. Find the total cost C of the same trip if a 15% tip is added.

(2)_____

3. A car rental service charges eight dollars a day and five cents a mile to rent a car. Find the expression for total cost C, in dollars, of renting a car for D days to travel M miles.

(3)_____

4. Find the total cost of the same trip as in question 3 if a sales tax of P percent is applied.

(4)_____

5. An English penny is currently worth 1.25 cents, lets say. There are 12 pence to a shilling and 20 shillings to a pound. Find the expression for dollars D if you have P pounds, S shillings and C Pence.

(5)_____

6. Find the expression for dollars in question 5 if the pound is devalued by 25%.

(6)_____

7. A carton contains S spools of thread. F feet of thread are wound upon each spool. Write an algebraic formula for T, the total number of yards of thread contained in the carton.

(7)_____

8. What is the formula for W, the number of feet of thread contained in a stockroom housing C such cartons as described in question 7.

(8)_____

Pre-test 3: Permutations Test

Suppose you are making license plates for a certain town. Each plate is four digits long and contains one each of the digits 1, 2, 3, and 4. List all the possible arrangements of 1234.

1234

Pre-test 4: Card Test

1. Suppose I have a deck of four cards containing 2 red cards and 2 black cards. I take the top card from the deck and place it (face up) on the table. It is a red card. Then I take the next card and put it on the bottom of the deck without determining what it is. I place the third card on the table. It is a black card. The following card I put undetermined below the others; while the next card, which is red, I put on the table. The procedure of alternately putting one card on the table and one on the bottom of the deck is continued until all the cards of the deck are placed on the table. The cards on the table appeared in this order: Red, Black, Red, Black. What was the order of the original deck?

_____ _____ _____ _____ (Put R or B in each space)
Top Bottom

2. Suppose I had a deck with 6 cards, half red and half black, and alternately placed one on the table and one on the bottom of the deck as above. If the cards appeared on the table in the order R,B,R,B,R,B, what was the order of the original deck?

_____ _____ _____ _____ _____ _____ (Put R or B in each space)
Top Bottom

3. Suppose I had a deck of 8 cards, half red and half black, and I alternately placed one on the table and one on the bottom of the deck as above. If the cards appeared on the table in the order R,B,R,B,R,B,R,B, what was the order of the original deck?

_____ _____ _____ _____ _____ _____ _____ _____ (Put R or B in each space)

Top Bottom

4. Suppose I had a deck of 8 cards, the ace through the 8 of clubs, and I alternately placed one on the table and one on the bottom of the deck as above. If the cards appeared on the table in the order A,2,3,4,5,6,7,8, what was the order of the original deck?

_____ _____ _____ _____ _____ _____ _____ _____ (Put A,1,2,3,4,5,6,7, or 8 in each space)

Top Bottom

INDIANA MATHEMATICAL PSYCHOLOGY PROGRAM

REPORT SERIES

(continued from back cover)

Report No.

72-1 *Castellan, N. J., Jr.*
Multiple-cue Probability Learning with Irrelevant Cues. Organizational Behavior and Human Performance, 1973, 9, 16-29.

72-2 *Castellan, N. J., Jr.*
The Effect of Different Types of Feedback in Multiple-Cue Probability Learning. Organizational Behavior and Human Performance, 1974, 11, 44-64.

72-3 *Levison, M. J. and Restle, F.*
Learning Redundant Solutions in Simple Concept Identification.

72-4 *Gardner, G. T.*
Evidence for Independent Parallel Channels in Tachistoscopic Perception.

*72-5 *Shiffrin, R. M.*
Information Persistence in Short-term Memory. Journal of Experimental Psychology, 1973, 100, No. 1, 39-49.

*72-6 *Lindman, H. R.*
Nonparametric Statistics, Bayesian and Classical: I. Sign Test and Mann-Whitney U Test.

72-7 *Restle, F.*
Serial Pattern Learning: Higher-Order Transitions. Journal of Experimental Psychology, 1973, 99, 61-69.

72-8 *Shiffrin, R. M., Craig, J. C., and Cohen, U.*
On the Degree of Attention and Capacity Limitations in Tactile Processing. Perception and Psychophysics, 1973, 13, 328-336.

72-9 *Shiffrin, R. M. and Schneider, W.*
An Expectancy Model for Memory Scanning.

*72-10 *Shiffrin, R. M. and Geisler, W. S.*
Visual Recognition in a Theory of Information Processing. Solso, R. The Loyola Symposium: Contemporary Viewpoints in Cognitive Psychology. Washington: Winston, 1973.

72-11 *Shiffrin, R. M., Gardner, G. T. and Allmeyer, D. H.*
On the Degree of Attention and Capacity Limitations in Visual Processing. Perception & Psychophysics, 1973, 14, 231-236.

72-12 *Levison, M. J. and Restle, F.*
Blank Trial Assumption Fails Stringent Test: The Effect of Probe Trials on Identification Problems with Redundant Relevant Cue Solutions. Also appears as Effects of Blank-Trial Probes on Concept-Identification Problems With Redundant Relevant Cue Solutions, Journal of Experimental Psychology, 1973, 98, 368-374.

72-13 *Castellan, N. J., Jr. and Edgell, S. E.*
An Hypothesis Generation Model for Judgment in Non-Metric Multiple-Cue Probability Learning. Journal of Mathematical Psychology, 1973, 10, 204-222.

72-14 *Jenkins, R.*
Deprivation Conditions in Multiple-Cue Probability Learning.

72-15 *Scholz, K. W.*
A Cognitive Approach to the Storage and Retrieval of Structured Verbal Material.

INDIANA MATHEMATICAL PSYCHOLOGY PROGRAM

REPORT SERIES

(continued from back cover)

72-16 *Edgell, S. E. and Castellan, N. J., Jr.*
The Configural Effect in Multiple-Cue Probability Learning. Journal of Experimental Psychology, 1973, 100, 310-314.

72-17 *Milech, Dan*
Task-Free Transfer: The Similarity of Serial Patterns which can be Encoded Hierarchically.

73-1 *Edgell, S. E., Geisler, W. S., III, and Zinnes, J. L.*
A Note on a Paper by Rumelhart and Greeno. Journal of Mathematical Psychology, 1973, 10, 86-90.

73-2 *6th Indiana Theoretical and Mathematical Psychology Conference*
Models for Information Processing. April 18-20, 1973. Indiana University. Abstracts and Papers presented.

73-3 *Shiffrin, R. M., Pisoni, D. B., and Castaneda-Mendez, K.*
Is Attention Shared Between the Ears? Cognitive Psychology, 1974 6, 190-215.

*73-4 *Shiffrin, R. M.*
The Locus and Role of Attention in Memory Systems. Presented at Attention and Performance V., Stockholm, Sweden, July, 1973.

73-5 *Glanzman, D. L. and Pisoni, David B.*
On Verifying Semantic Inferences.

73-6 *Lindman, Harold R.*
A Numerical Approximation to Ordered Metric Scales.

73-7 *Durding, Bruce M.*
The Perceptual Organization of a Polyrhythm.

73-8 *Shiffrin, R. M. and Grantham, D. W.*
Can attention be allocated to sensory modalities?

73-9 *Castellan, N. J., Jr.*
Laboratory programming languages and standardization. Behavioral Research Methods and Instrumentation, 1973, 5, 249-252.

73-10 *Zagorski, M.*
A Topological Test of Metric Models of Stimulus Similarity.

*73-11 *Zinnes, J. L. and Griggs, R. A.*
Probabilistic, Multidimensional Unfolding Analysis.

74-1 *Picek, J. S., Sherman, S. J. and Shiffrin, R. M.*
Cognitive Organization and Coding of Social Structures.

74-2 *Griggs, R. A.*
Constancy Scaling Theory and Mueller-Lyer Illusion: More Disconfirming Evidence.

74-3 *Griggs, R. A.*
Overestimation of Lower Vertical in a Comparative Length Judgment Task.

74-4 *Edgell, S. E.*
Configural Information Processing in Decision Making.

74-5 *Forshee, Jerry C.*
Addition and Subtraction: One Cognitive Process or Two?

74-6 *Durding, Bruce M.*
The Organization of Spatial and Temporal Context in Human Memory

INDIANA MATHEMATICAL PSYCHOLOGY PROGRAM

REPORT SERIES

(continued from back cover)

Report No.

74-7 *Griggs, Richard A.*
Logical Errors in Comprehending Set Inclusion Relations in Meaningful Text

74-8 *Kalme, Charles I.*
The Basic Search Routine for Selecting a Move in Chess

74-9 *Kalme, Charles I.*
A Pattern Oriented Encoding of Chess Knowledge

75-1 *Mayer, Richard E.*
Instructional Variables in Meaningful Learning of Computer Programming

INDIANA MATHEMATICAL PSYCHOLOGY PROGRAM

REPORT SERIES

Report No.

*69-1 ZINNES, D. A., ZINNES, J. L., & MCCLURE, R. D., Hostility in communications: A study of the 1914 crisis. In C. F. Herman (Ed.), *Decision Making in International Crisis*. New York: Academic Press, 1970.

*69-2 LINDMAN, H. R., A numerical approximation to ordered metric scales.

69-3 CASTELLAN, N. J., Jr., Determination of joint distributions from marginal distributions in dichotomous systems. *Psychometrika*, 1970, **35**, 439-454.

*69-4 SWINNEY, D. A., Recognition search through short term memory in aphasics.

69-5 RESTLE, F., Speed of adding and comparing numbers. *Journal of Experimental Psychology*, 1970, **83**, 274-278.

†69-6 MERRYMAN, C. T., Effects of strategies on associative symmetry. (Order No. 70-10,270.)

*69-7 SHIFFRIN, R. M., Memory search. In D. A. Norman (Ed.), *Models of Memory*. New York: Academic Press, 1970, 375-447.

*69-8 SHIFFRIN, R. M., The temporal dimension in the memory search.

*69-9 RESTLE, F., Rapid judgments of numbers: Half magnitudes.

69-10 GILLMAN, C. B., The effects of interstimulus interval and feedback on the time order effect in judgments of numerousness and line length.

69-11 CASTELLAN, N. J., Jr., A note on the expected value of a coefficient of reproducibility. *Multivariate Behavioral Research*, 1969, **4**, 363-370.

70-1 RESTLE, F., Theory of serial pattern learning: Structural trees. *Psychological Review*, 1970, **77**, 481-495.

70-2 LINDMAN, H. R., Inconsistent preferences among gambles. *Journal of Experimental Psychology*, 1971, **89**, 390-397.

70-3 SHIFFRIN, R. M., Delayed free recall and theories of forgetting. (A shorter version has been published as SHIFFRIN, R. M., Forgetting: Trace erosion or retrieval failure? *Science*, 1970, **168**, 1601-1603.)

70-4 RESTLE, F., & BROWN, E., Organization of serial pattern learning. In G. H. Bower (Ed.), *Psychology of Learning and Motivation*, Vol. 4. New York: Academic Press, 1970, 249-331.

70-5 BROWN, E., & GORDON, R., Internal vs. external control as related to performance in two-choice probability learning. *Organizational Behavior and Human Performance*, 1971. **6**, 200-214.

Report No.

70-6 POTTS, G. R., & SHIFFRIN, R. M., Repetitions, blank trials, and the vonRestorff effect in free recall memory. *Journal of Exeprimental Psychology*, 1970, **86**, 128-130.

70-7 BURNSIDE, W., Judgment of short time intervals while performing mathematical tasks. *Perception & Psychophysics*, 1971, **9**, 404-406.

70-8 GODWIN, W. F., Subgroup pressures in small-group consensus processes.

70-9 POTTS, G. R., Spacing of item repetitions in verbal learning paradigms. (An expanded version of this paper is contained in POTTS, G. R., Distance from a massed double-presentation or blank trial as a factor in paired associate list learning. *Journal of Verbal Learning and Verbal Behavior*, 1972, **11**, 375-386.)

70-10 YOST, W. A., Tone-on-tone binaural masking.

70-11 ZINNES, J. L., ZINNES, D. A., & WILKENFELD, J., A Markovian analysis of hostility in the 1914 crisis.

71-1 CASTELLAN, N. J., Jr., Comments on the "Lens Model" equation and the analysis of multiple-cue judgment tasks. *Psychometrika*, 1973, **38**, (in press).

71-2 RESTLE, F., Serial patterns: The role of phrasing. *Journal of Experimental Psychology*, 1972, **92**, 385-390.

†71-3 BROWN, E., Abstraction and hierarchical organization in the learning of periodic sequences. (Order No. 72-6752.)

71-4 RESTLE, F., & BURNSIDE, B. L., Tracking of serial patterns. *Journal of Experimental Phychology*, 1972, **95**, No. 1 (in press).

71-5 SHAFFER, W. O., & SHIFFRIN, R. M., Rehearsal and storage of visual information. *Journal of Experimental Psychology*, 1972, **92**, 292-296.

71-6 CASTELLAN, N. J., Jr., The analysis of multiple criteria in multiple-cue judgment tasks. *Organizational Behavior and Human Performance*, 1972, **8**, 242-261.

71-7 POTTS, G. R., A cognitive approach to the encoding of meaningful verbal material.

71-8 SCHOLZ, K. W., Computerized process control in behavioral science research. *Behavior Research Methods and Instrumentation*, 1972, **4**, 203-208.

Report listing contigued on inside pages.

*These reports are no longer available from our office.

†These reports may be ordered at the prices indicated from University Microfilms, A Xerox Company, Dissertation Copies, P.O. Box 1764, Ann Arbor, Michigan, 48106. (Microfilm $4; Xerox $10.)